versie: 4 februari 2011

# RAAK MKB Future Store Virtual Dressing Room

## Ontwerp en implementatie


**Auteur**

J.K. van Kruijssen
cardamon@gmail.com

**Onder supervisie van**

E.H.A. Jannink
e.h.a.jannink@saxion.nl

Onderzoeksminor
Omgevingsintelligentie en -Interactie

**In samenwerking met**

# Inhoudsopgave


I Ontwerp 4

1 Opstelling 5

2 Low-level feature extraction 7

3 Mid-level feature extraction 13

II Implementatie 17

4 Requirements 18

5 Realisatie 19

III Appendices 27

A Wiskundige begrippen en notaties 28

B Bibliography 31

# Introductie

## Document

Dit document bevat het ontwerp van de software en een beschrijving van de te gebruiken hardware. Elke feature in het systeem heeft een eigen hoofdstuk waarin de grondbeginselen en de gebruikte algoritmes worden uitgelegd.

## Grondbeginselen

De grondbeginselen in dit document kunnen door programmeurs met een gemiddelde wiskunde achtergrond worden gelezen zonder (veel?) gebruik te maken van handboeken. In een bijlage worden een aantal begrippen en notaties uitgelegd.

# Deel I

# Ontwerp

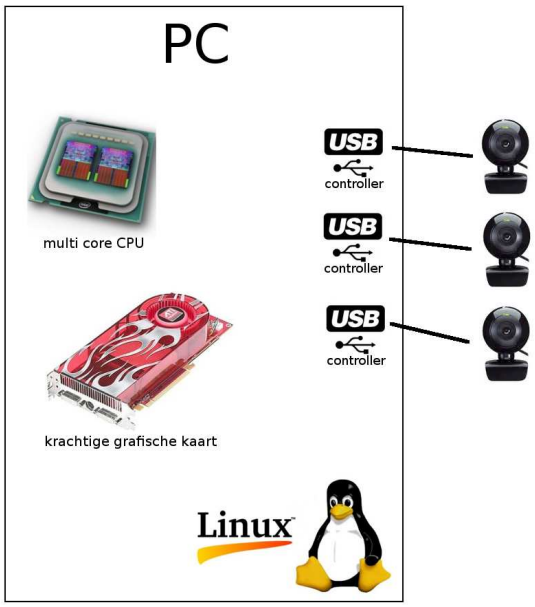
## Hoofdstuk 1

# Opstelling

*In dit hoofdstuk wordt de gebruikte hardware en diens opstelling beschreven.*

## 1.1 Hardware

De opstelling is bestaat uit de volgende onderdelen.




Figuur 1.1: De opstelling van het prototype.

- Drie keer een ‘Logitech Webcam C12’.
- Een Linux PC met (o.a.):
  - drie USB controllers.

- een grafische kaart welke een goede performance heeft OpenGL hardware acceleratie.
- een multi-core CPU.

## 1.2 Software

Voor een overzicht van de gebruikte bibliotheken en het compileren en gebruik van van het prototype, zie hoofdstuk 5.1 en 5.2.

## 1.3 Camera's gebruiken

De Logitech Webcam C12 is weliswaar geschikt voor het prototype, maar moet eerst handmatig worden ingesteld om een aantal 'eigenaardigheden' te voorkomen. Voor elke camera, doe het volgende:

```
# vlc v4l2:///dev/video1 &
# v4l2ucp /dev/video1 &
```

waarbij `video1` dus ook `video0` of `video2` kan zijn. Stel vervolgens op het controle paneel in dan alle 'automatische' features uit staan. Zet eerst de 'Exposure, Auto Priority' en 'White Balance Temperature, Auto' uit, en selecteer vervolgens uit het drop down menu van 'Exposure, Auto' de optie 'Manual Mode'.

## Hoofdstuk 2

# Low-level feature extraction

*In dit hoofdstuk worden de grondbeginselen en algoritmes van low-level feature extraction beschreven. Low-level feature extraction is de eerste stap in de analyse van de camera feeds: bewegende onderdelen in een scene worden per camera geïdentificeerd.*

## 2.1 Ontwerpkeuzes

**Alternatieven LLFE** De mogelijkheden voor low-level feature extraction zijn bijvoorbeeld (Turaga et al. [2008]) als volgt in te delen.

- Algoritmes die 'ogenschijnlijke bewegingen' van pixels detecteren (een pixels met bepaalde componenten bevindt zich op tijdstip $t$ op punt $(a, b)$ en op tijdstip $t+1$ op punt $(a+x, b+y)$.
- Zoals het voorgaande, maar dan niet voor individuele pixels, maar voor 'objecten' (groepen van meerdere pixels).
- Background subtraction, waarbij de achtergrond van de scene wordt verwijderd om zo silhouettes van de voorgrond te creëren.
- Technieken gebaseerd op temporal en/of spational filters, bijvoorbeeld het indelen van pixels die gedurende een bepaalde tijd niet veranderen bij de achtergrond.

**Keuze** Gekozen is voor background subtraction, omdat deze methode het meest wordt gebruikt door wetenschappers die in een soortgelijke context werken (zie verdiepende literatuurstudie).

**Alternatieven voor background subtraction** In Toyama et al. [1999] worden elf alternatieven genoemd en met elkaar vergeleken. Voorbeelden zijn:

- Adjacent Frame Difference, waarbij simpelweg het verschil tussen twee frames wordt bekeken
- Mean & Covariance, waarbij een grenswaarde van de Mahalanobis afstand die 'huidige' pixels hebben tot een achtergrond model wordt gebruikt om een opdeling te maken in voor- en achtergrond.
- ...

**Keuze** Gekozen is voor Mean & Covariance (althans, een variant hiervan). Deze methode is herhaaldelijk gebruikt binnen eenzelfde context (zie verdiepede literatuurstudie). Argumenten die voor deze methode spreken worden genoemd, en zijn de volgende:

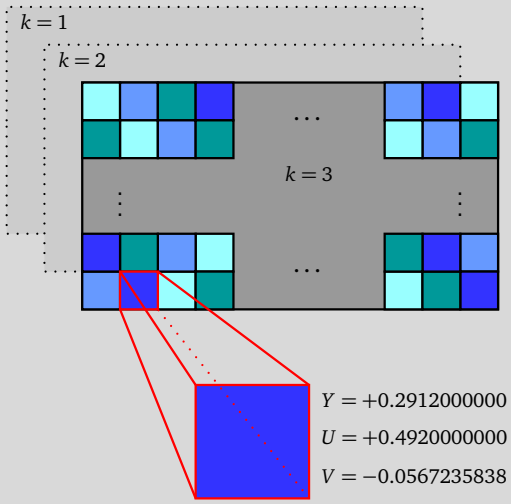
- Deze methode is relatief snel: ze vereist niet zoveel rekenkracht als andere methodes.
- De resultaten zijn weliswaar niet zo nauwkeurig als andere methodes, maar dit is ook geen vereiste omdat uiteindelijk meerdere camera's gebruikt gaan worden.
- Andere methodes die relatief gezien nog sneller zijn, zijn te onnauwkeurig.

De keuze wordt dus gekenmerkt door een afweging van de nauwkeurigheid tegen de complexiteit van de methode.

## 2.2 Background subtraction

### 2.2.1 Pixels als vectoren

We beschouwen een enkele frame, opgenomen door een camera. Deze frame is een afbeelding, met afmeetingen breedte $\times$ hoogte. $(x, y)$ is de coördinaat van een pixel: de $x$-de pixel van links en de $y$-de pixel van de bovenkant. Elke pixel heeft een Y-, een U- en een V-component. Elke frame van een camera wordt in dit document genummerd met $k$. De eerste frame die wordt opgenomen door de camera heeft nummer $k = 1$, de tweede $k = 2$, etc.




Figuur 2.1: Frames van een camera met een uitvergroting van een pixel met daarnaast de waardes van de componenten.

Elke pixel in een frame kan worden voorgesteld door een vectorfunctie $\overrightarrow{C}$ van de drie variabelen $x$, $y$, en $k$:

$$\overrightarrow{C} : \mathbb{Z}^3 \to \mathbb{V}^3$$

waarbij de elementen van de vector de YUV componenten van de pixel zijn. YUV componenten zijn gerelateerd aan RGB componenten, deze relatie is verderop te lezen.

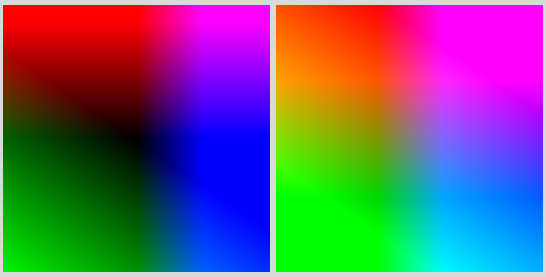
Als de frames van figuur 2.2.1 een grootte zouden hebben van $640 \times 480$, dan zou de uitvergrote pixel de volgende vector zijn:

$$\overrightarrow{C_{2,479}(3)} = \begin{bmatrix} 0.2912000000 \\ 0.4920000000 \\ 0.0567235838 \end{bmatrix}$$

namelijk de pixel op locatie $(2, 479)$ van het 3-de frame.

### 2.2.2 Berekenen van YUV componenten

YUV is net zoals RGB een kleurenruimte. De Y component geeft de helderheid, terwijl de U en V componenten de kleur bepalen.



Figuur 2.2: Het UV-plane bij $Y = 0$ en $Y = 0.5$

Als de YUV componenten niet direct kunnen worden opgevraagd aan de camera, kunnen deze worden berekend aan de hand van de RGB componenten met een enkele matrixvermenigvuldiging (gekopiëerd uit wikipedia):

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

### 2.2.3 Achtergrond leren

Low-level feature extraction begint met het 'leren' van de achtergrond. De eerste stap van dit leerproces is het bepalen van gemiddelde waardes voor elke pixel van een aantal frames (bijvoorbeeld 50).


Figuur 2.3: Voorbeeld van segmentatie van voor- en achtergrond door Guan.

Tijdens het leerproces is er geen actor in de scene. De frames bestaan dus alleen uit 'achtergrondbeeld'. Hierbij is $\overrightarrow{C(x,y,k)}$ de huidige pixel op locatie $(x,y)$ van frame nummer $k$. Een gemiddelde pixel $\overrightarrow{G}$ kan worden gedefinieerd als volgt:

$$\overrightarrow{G(x,y,k)} \stackrel{\text{def}}{=} \frac{1}{k}\sum_{i=1}^{k}\overrightarrow{C(x,y,i)}$$

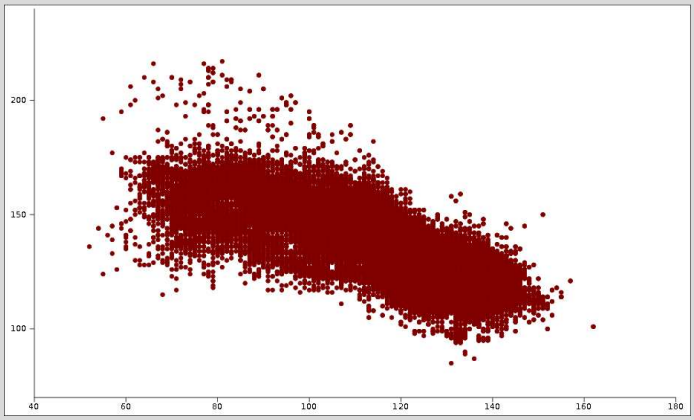
Na een bepaalde tijd (i.e. voor een bepaalde $k=m$), kan een achtergrondpixel worden vastgesteld als volgt:

$$\overrightarrow{B(x,y)} \stackrel{\text{def}}{=} \overrightarrow{G(x,y,m)}$$

waarbij de geschikte waarde voor $m$ experimenteel wordt bepaald, en waarschijnlijk zo rond de 50 zal liggen.

### 2.2.4 Covariantie tussen de YUV componenten van een pixel

De verschillende componenten van de pixels zijn aan elkaar gerelateerd. Zie de onderstaande figuur voor een voorbeeld. Deze onderlinge relatie kan worden weergegeven in een covariantiematrix.




Figuur 2.4: De correlatie tussen de U (horizontale as) en V (verticale as) componenten van alle pixels van een enkele frame

$Y_C$, $U_C$, $V_C$ zijn de verzamelingen van alle Y, U en V componenten van alle $C$:

$$Y_C \stackrel{\text{def}}{=} \{0 < k \leq m, 0 \leq x < \text{width}, 0 \leq y < \text{height} : C(x,y,k)_Y\}$$

waarbij $U_C$, $V_C$ soortgelijke definities hebben. De covariantiematrix matrix van de achtergrondpixel, $\mathbf{CV_C}$, kan nu als volgt worden berekend:

$$\mathbf{CV_C} = \begin{bmatrix} \text{cov}(Y_C, Y_C) & \text{cov}(Y_C, U_C) & \text{cov}(Y_C, V_C) \\ \text{cov}(U_C, Y_C) & \text{cov}(U_C, U_C) & \text{cov}(U_C, V_C) \\ \text{cov}(V_C, Y_C) & \text{cov}(V_C, U_C) & \text{cov}(V_C, V_C) \end{bmatrix}$$

waarbij $\text{cov}(P,Q)$ de covariantie tussen de verzamelingen $P$ en $Q$ voorstelt, en als volgt is gedefinieerd (gebruikmakend van de eerder gedefinieerde $m$):

$$\text{cov}(P,Q) = \frac{1}{m-1} \sum_{i=1}^{m} (P_i - \overline{p})\,(Q_i - \overline{q})$$

waarbij $\overline{p}$ en $\overline{q}$ de gemiddelde waardes van $P$ en $Q$ zijn. $P_i$ en $Q_i$ zijn de $i$-de elementen van $P$ and $Q$. De gemiddelde waardes van $Y_A$, $U_A$ en $V_A$ zijn reeds beschikbaar: dit zijn de componenten van $\overrightarrow{B(x,y)}$.

Nu kan een model van de achtergrond $\text{MB}_{x,y}$ informeel worden gedefinieerd als een combinatie van $\overrightarrow{B(x,y)}$ en $\mathbf{CV_C}$.

### 2.2.5 Mahalanobis afstand

Zodra het model van de achtergrond is vastgesteld kan de actor in de scene stappen (zie tweede afbeelding van figuur 2.2.3.

$\overrightarrow{C(x,y)}$ stelt nu een pixel voor uit een frame van de scene waar de actor te zien is. De Mahalanobis afstand (althans, het kwadraat hiervan) tussen elke $\overrightarrow{C(x,y)}$ en $\mathrm{MB}_{x,y}$ is als volgt gedefinieerd:

$$D^2\left(\overrightarrow{C(x,y)}, \mathrm{MB}_{x,y}\right) \stackrel{\text{def}}{=} \left(\overrightarrow{C(x,y)} - \overrightarrow{B(x,y)}\right)^T \mathbf{CV_C}^{-1} \left(\overrightarrow{C(x,y)} - \overrightarrow{B(x,y)}\right)$$

Van dit getal wordt niet de wortel berekend omdat we alleen geïnteresseerd zijn in de relatie van $D^2$ met een drempelwaarde $r$. Deze drempelwaarde zal experimenteel worden bepaald, of een parameter van het programma worden.

## 2.3 Algoritmes

### 2.3.1 Achtergrond leren

Pseudo code voor het leren van frames:

```
for each pixel in a frame
  set Ys, Us, Vs; // sets
  int y, u, v;    // sums

  for each of the 50 frames learned
    append the Y, U, V components to the sets;
    add the Y, U, V, components to the sums;
  end

  calculate mean Y, U, V using the sums;
  calculate inverse covariance matrix;
end
```

### 2.3.2 Segmentatie maken

Pseudo code voor het opdelen van een frame in voor- en achtergrond:

```
while(true)
  for each pixel in the current frame
    calculate difference between current and mean pixel;
    calculate square mahalanobis distance using the
      difference and the inverse covariance matrix;

    if (sq. mah. distance > THRESHOLD)
      pixel in foreground
    else
      pixel in background
  end
end
```

## Hoofdstuk 3

# Mid-level feature extraction

*Mid-level feature extraction betreft de 3D reconstructie op basis van het resultaat van low-level feature extraction. Let op: dit gedeelte is onvoltooid, en dus slecthts gedeeltelijk ontworpen.*

## 3.1 Ontwerpkeuzes

**Alternatieven MLFE** Welke methodes voor 3D reconstructie geschikt zijn wordt beperkt door de gemaakte keuze voor LLFE. Daar is gekozen voor het aanmaken van silhouetten, 3D reconstructie kan dan als volgt (o.a. Hasenfratz et al. [2003], Michoud et al. [2007]):

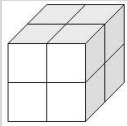
- Een 'volumterische' aanpak, waarbij een voxel space wordt opgebouwd.
- Een 'oppervlakte' aanpak, waarbij direct mesh wordt geconstrueerd.

In de periode (ongeveer) 2000 – 2008 kiezen wetenschappers meestal voor de volumetrische aanpak. Recentlijk wordt meer voor een oppervlakte aanpak gekozen.

**Keuze** Gekozen voor dit project is de volumetrische aanpak, omdat deze eenvoudiger te realiseren is. De nieuwere methodes (zie bijv. Petit et al. [2010]) geven betere resultaten; een dergelijke methode kan wellicht in de toekomst als verbeterslag worden geïmplementeerd.

## 3.2 Een voxel space met slices

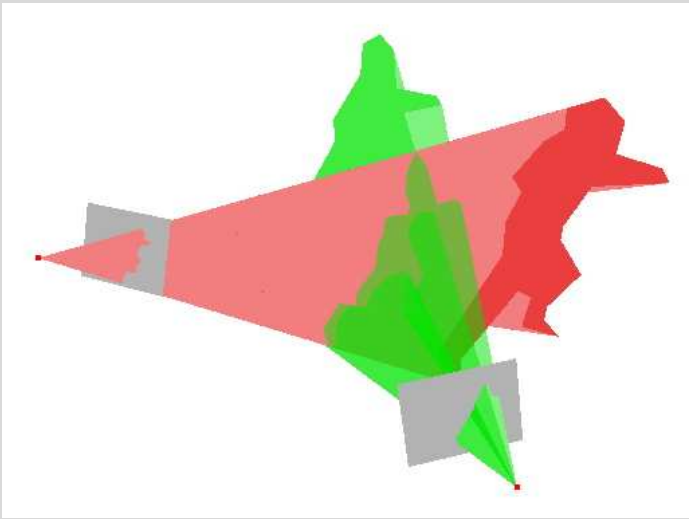
De voxel space kan worden gezien als een geometrische balk die wordt gesneden in verschillende slices. Het voorvlak van de balk is de eerste slice. Elke slice heeft in de datastructuur slechts twee dimensies: de breedte en hoogte. De 'dikte' van een slice wordt bepaald door de grootte van een voxel (de resolutie). Een voxel space bestaat dus uit $d$ slices met elk een afmeting van $w \cdot h$ ($d$, $w$, en $h$ zijn respectievelijk de diepte, breedte en hoogte).



Figuur 3.1: Een mogelijke manier om een $2 \times 2 \times 2$ voxel space te visualiseren.

## 3.3 Visual hull

De silhouetten worden gecombineerd tot een 3D reconstructie door deze te projecteren in een voxel space. Daar waar de projecties elkaar snijden bevindt zich de visual hull. De visual hull is het maximaal omsluitende volume van een object. Het kan dus zo zijn dat de visual hull voxels bevat welke geen onderdeel zijn van het daadwerkelijke object, maar omgekeerd kan het niet voorkomen dat delen van het object niet worden gerepresenteerd.
Voxels die wel onderdeel vormen van de visual hull en niet van het object worden vooral gevormd in regio's van het object die concaaf zijn. Des te meer camera's worden te gebruikt met verschillende invalshoeken, des te minder van deze onwenselijke voxels.

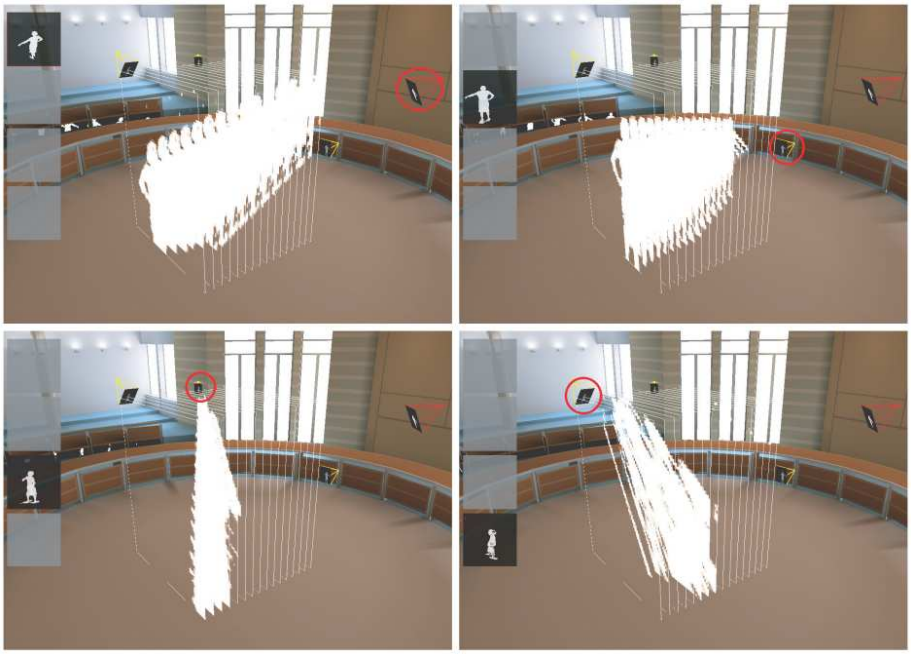


Figuur 3.2: Silhouette cones (VHW) op basis van twee camera's.

In een voxel space die bestaat uit slices moet dus op elke slice een gedeelte van de silhouette cones worden gerendered. Dit moet zodanig dat uiteindelijk een voxel alleen 'actief' is als deze in de silhouette cone zit van alle aanwezige camera's.

Stel een camera staat loodrecht op het voorvlak van een voxel space, zonder enige draaiing. De silhouette cone doorsnijdt dan elke slice, waarbij op de voorste slice een 'klein' silhouette wordt geprojecteerd, en op de achterste slice een 'relatief groter' silhouette.

Stel dat nu een camera wordt toegevoegd die in relatief gezien 'naast' het object staat, en hoek van 90 graden maakt met de silhouette cone van de eerste camera. Op elke slice moet dan de silhouette niet alleen van de ene zijkant naar de andere zijkant steeds groter worden, maar ook worden gedraaid, zodat deze dezelfde oriëntatie heeft.



Figuur 3.3: Een projectie van silhouette cones in een voxel ruimte op basis van slices door Hasenfratz et al. [2003].

Voor elke slice moet dus worden bepaald, op basis van de parameters van de camera (roll, pitch, yaw, straal tot het middelpunt tot de scene):

- de perspectief correctie
- de draaiing van het silhouette

## 3.4 Algoritmes

Het volgende algoritme is ten tijde van schrijven gedeeltelijk gerealiseerd (zie Implementatie).

### 3.4.1 Combinatie van silhouetten op de GPU

Zie ook Hasenfratz et al. [2003], pp. 5. Pseudo code voor het opbouwen van een voxel space waarin een 3D reconstructie op basis van de combinatie van silhouetten plaats vindt:

```
while(true)
  disable blending;
  for each camera
    move silhouette to GPU memory;
    set the texture matrix to the camera's parameters;
    for each slice of the voxel space
      set location to slice;
      render a quad with the silhouette;
    end
    enable blending;
  end
  read result from GPU;
  convert result to a voxel space;
end
```

Blending zorgt er voor dat alleen wordt gerendered als een pixel in een slice voorkomt in elk silhouette. Dit gebeurt door een pixel op zwart te zetten als deze niet al gerendered is in de vorige iteratie van de loop EN niet zal worden gerendered in de huidige iteratie van de loop.

Het verplaatsen van de silhouette naar het geheugen van de GPU kan door deze in te lezen als texture.

Zie voor meer details over het instellen van de texture matrix op parameters van de camera's het volgende gedeelte van dit document (Implementatie).

Het resultaat moet worden gerendered en uitlezen, maar ook zal de GPU worden gebruikt voor het renderen van andere zaken (de voxel space, of toekomstige onderdelen van het project). Daarom kan voor deze stap het beste een off-screen frame buffer worden gebruikt.

# Deel II

# Implementatie

Hoofdstuk 4

# Requirements

*De eisen aan het systeem. Sommige eisen zijn door de beperkte hoeveelheid tijd slechts gedeeltelijk gerealiseerd.*

## 4.1 Abstract requirements

Het systeem zal...

R1 de mogelijkheid bieden om meerdere camera's aan te sturen en de frames die deze camera's produceren te verwerken.

R2 een bepaald aantal frames leren om zo een achtergrondmodel op te stellen.

R3 voorafgaand hieraan een bepaald aantal frames over slaan zodat de camera zichzelf kan opstarten (scherpstellen, etc.).

R4 na het bepalen van een achtergrondmodel elke frame opdelen in een voor en achtergrond: pixels die 'dicht' in de buurt van het achtergrondmodel zitten horen bij de achtergrond, en pixel die hier 'ver' vanaf staan horen bij de voorgrond.

R5 de mogelijkheid bieden parameters (hoeken, positie) van de camera's in te stellen.

R6 verschillende silhouetten combineren tot een 3D reconstructie.

## 4.2 Quality Requirements

1. Het systeem zal in de vorm van een static library worden gerealiseerd.
2. De code zal zoveel mogelijk herbruikbaar zijn binnen een andere context.
3. De code zal zoveel mogelijk uitbereidbaar zijn.
4. Het systeem zal een dusdanige performance hebben dat deze werkt op consumenten hardware.

Hoofdstuk 5

# Realisatie

## 5.1 Overzicht van gebruikte libraries

De volgende libraries worden gebruikt:

**Armadillo** pakket voor met SIMD geoptimaliseerde lineaire algebra functies, zoals matrix vermenigvuldigingen.

**OpenGL, Freeglut** voor 2D/3D rendering functionaliteit, window en toetsenbord management, etc.

Deze kunnen (onder ubuntu) als volgt worden geïnstalleerd:

```
# sudo aptitude update && sudo aptitude install -y build-essential
libblas-dev liblapack-dev libboost-dev && sudo aptitude install -y
libarmadillo-dev freeglut3-dev
```

N.B.: In sommige versies van Ubuntu is de package van Armadillo niet functioneel. Als een compile error hier op wijst, installeer dan Armadillo handmatig. Verwijder eerst libarmadillo-dev en libarmadillo, download en volg de instructies op `http://arma.sourceforge.net/download.html`.

Voor het genereren van de documentatie is ook Doxygen vereist. Deze kan op Ubuntu worden geïnstalleerd als volgt:

```
# sudo aptitude update && sudo aptitude install -y doxygen
```

## 5.2 Compilatie van de source en documentatie

Compilatie van de source kan met de makefile in de `code/src` map:

```
# make
# make clean
```

Compilatie van de doxygen documentatie kan in de `code` map:

```
# doxygen Doxyfile
```

De huidige test van het prototype kan worden uitgevoerd door de binary in de `code/bin` map te starten. Zorg er voor dat er geen enkele beweging voor de camera plaats vindt, totdat op het scherm de 3D reconstructie verschijnt!

```
# ./main
```

Tijdens het uitvoeren van het programma kunnen de camera parameters worden aangepast. Deze parameters worden echter ten tijde van schrijven niet correct afgehandeld! Besturing is als volgt:

- Kies een toets 1-9 om een camera te kiezen.
- Kies y voor yaw, r voor roll, p voor pitch en z voor straal.
- Gebruik vervolgens e om de waarde te verhogen en d om de waarde te verlagen.

Bijvoorbeeld: kies 1, dan y dan e. Dan p, dan e. De yaw en pitch van camera 1 zijn nu lichtelijk verhoogd.

Momenteel wordt slechts een camera gebruikt. Om meer camera's toe te voegen, pas dan `test6.cpp` aan. Decomment / kopieer bijvoorbeeld de regel die begint met

```
mn->add_capture_device
```

en pas de meegegeven URI aan om de gewenste camera te gebruiken.

## 5.3 Overzicht van namespaces en classes

Hier volgt een kort overzicht van de implementatie aan de hand van de meest relevante classes. Een compleet overzicht is te vinden in de doxygen documentatie.

### *fscmn namespace*

In deze namespace zitten classes die de standaardfunctionaliteit van C++ hier en daar uitbereiden.

**executor** Deze klasse voert taken door deze te verspreiden over een aantal threads. Het aantal threads is typisch gelijk aan het aantal CPU cores. Op deze manier kan aan load balancing worden gedaan.

**task en task_complete_barrier** Bass class voor de taken die door de executor worden uitgevoerd en een barrier die een thread laat wachten totdat alle toegevoegde taken zijn uitgevoerd.

### *fscamctrl namespace*

In deze namespace zitten classes die de camera's aansturen en diens uitvoer afhandelen.

**capture_device en v4l2_capture_device** Classes die een enkele camera aansturen. Momenteel is alleen ondersteuning voor V4L2 (Video for Linux 2) devices. Een device kan worden geopend en gesloten, kan starten en stoppen met streamen en er kan een frame van worden gelezen.

**`capture_device_manager` en `frame_store`** Deze klasses maken het mogelijk meerdere capture devices tegelijk aan te sturen, en de frames die worden gegenereerd efficiënt af te handelen.

#### *`fsllfe` namespace*

In deze namespace zitten classes die low-level feature extraction realiseren.

**`frame_converter` en sub classes** Deze klasses transformeren een frame in iets anders. De `yuv422_rgba_frame_converter` converteert bijvoorbeeld YUV 4:2:2 frames naar RGB plaatjes welke op het scherm kunnen worden weergegeven. De `bglearn_frame_converter` leert de achtergrond van een frame, waarna deze overschakeld op een `bg_segment_frame_converter` die een frame opdeelt in een voor en achtergrond.

#### *`fsmlfe` namespace*

In deze namespace zitten classes die mid-level feature extraction (3D reconstructie) realiseren.

**`renderer` en sub class** Deze klassen kunnen bevatten de aansturing van OpenGL, zoals het aansturen van een off-screen framebuffer. `reconstruction_renderer` voegt daar nog functionaliteit voor 3D reconstructie aan toe, voor zover deze nu gerealiseerd is.

**`voxelspace`** Deze klasse representeert een voxel space opgebouwd uit slices. De voxel space kan in toekomstige onderdelen van het prototype worden gebruikt, of kan worden gerendered met een `renderer`.

## 5.4 Gebruik van LLFE

Gebruik van de LLFE gaat typisch als volgt:

```
fscamctrl::capture_device_manager *mn = NULL;
fsllfe::frame_store *fs = NULL;
fsllfe::converted_frame **dfr;

void init() {
  // create managers
  mn = new fsllfe::capture_device_manager();
  fs = new fsllfe::frame_store();

  // add some capture devices to the device manager
  mn->add_capture_device(
    "v4l2://yuv422@/dev/video0", // protocol, format, device name
    640, 480,                    // preferred width and height
    0.0f,                        // z offset
    0.0f, 0.0f, 0.0f             // roll,pitch,yaw
  );
```

```
  // add all capture devices to the frame store
  VECT_ITERATE(fscamctrl::capture_device *, it, mn)
    fs->add_source(mn->get_source_frame(*it));

  // storage for pointer to dest frames
  dfr = new fsllfe::converted_frame * [mn->size()];

  // attach the converter(s) we want to use
  fs->attach_to_all(fsllfe::yuv422_bglearn_frame_converter::↘
    →instance());

  // begin streaming
  mn->start_streaming();
}

void deinit() {
  if (mn != NULL) {
    mn->stop_streaming();
    delete mn;
  }

  delete fs;
  delete bm;

  fsmlfe::voxelspace::clean();
  fsmlfe::reconstruction_renderer::clean();

  delete [] dfr;
}

// this function should be called in something like a
// while(true) loop
void mainloop() {
  // read frames and convert
  mn->refresh_frames();
  fs->convert_all();

  // for each device ...
  unsigned int i = 0;
  VECT_ITERATE(fsllfe::capture_device *, it1, mn) {
    dfr[i] = fs->get_dest_frames(mn->get_source_frame(*it1));

    // for each converted frame ...
    VECT_ITERATE(fsllfe::dest_frame *, it2, dfr[i]) {
      // do something usefull with the
      // converted frame in (*it2)
    }
    ++i;
  }
```

```
}
```

## 5.5 Configuratie van LLFE

Een aantal magic numbers zijn van belang, en kunnen naar omstandigheden worden aangepast.

**LEARN_DELAY** Het aantal frames dat moet worden gewacht voordat met het background learnen wordt begonnen. Momenteel 30.

**LEARN_FRAMES** Het aantal frames dat wordt gelearned. Momenteel 30.

**MAHALANOBIS_THRESHOLD** De waarde voor de in het ontwerp genoemde threshold voor de mahalanobis afstand. Momenteel 100.0f.

## 5.6 Gebruik van MLFE

Gebruik van de MLFE gaat typisch als volgt. Zie ook gebruik van LLFE, en zie het onderstaande als een uitbereiding / gedeeltelijke vervanging.

**NOTA BENE:** MLFE bevat tijde van schrijven nog een aantal TODO's. Zie hiervoor de volgende sectie.

```
#define rdr fsmlfe::reconstruction_renderer::instance()
#define vxl fsmlfe::voxelspace::instance()

GLuint *silhouette_texids;
fscamctrl::cam_external_params **params;

// add this to the init
void addthistoinit() {
  // generate texture ids for the silhouettes
  silhouette_texids = new GLuint[mn->size()];
  glGenTextures(mn->size(), silhouette_texids);

  // storage for camera params
  params = new fscamctrl::cam_external_params * [mn->size()];
}

// add this to the deinit
void addthistodeinit() {
  delete silhouette_texids;
  delete [] params;
}

// this function should be called by something
// like GLUT's display callback
void displayfunc() {
  // read frames and convert
  mn->refresh_frames();
```

```
  fs->convert_all();

  // get frames and parameters
  unsigned int i = 0;
  VECT_ITERATE(fscamctrl::capture_device *, it, mn) {
    dfr[i] = fs->get_converted_frames(mn->get_source_frame(*it))->↘
      →front();
    params[i] = mn->get_external_params(*it);
    ++i;
  }

  // create silhouettes
  rdr->create_silhouettes(mn->size(), silhouette_texids, dfr);

  // create slices, read slices into voxelcube
  rdr->go_offscreen();
  rdr->clear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT, 0.0f, 0.0f,↘
    → 0.0f, 0.0f);
  rdr->render_slices(mn->size(), silhouette_texids, dfr, params, ↘
    →vxl->get_depth());
  rdr->go_onscreen();

  // create voxel space
  vxl->create_from(rdr);

  // render voxelspace
  rdr->clear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT, 0.2f, 0.2f,↘
    → 0.7f, 0.0f);
  rdr->render_voxelspace(vxl);

  glutSwapBuffers();
}
```

## 5.7 MLFE nog niet voltooid

### 5.7.1 TODO's

Het MLFE gedeelte is nog niet voltooid. Momenteel worden de camera parameters niet correct meegenomen in de projectie. Een eventule volgende groep kan deze functionaliteit realiseren. Zie hiervoor de 'TODO's in `code/src/reconstruction.cpp`. Gezien het verloop van de implementatie is het goed mogelijk dat zodra dit issue is gefixed zich een nieuw probleem aandient, deze zijn uiteraard ten tijde van schrijven niet te voorspellen...

### 5.7.2 Texture matrix

Hier volgt nu een korte beschrijving van de het werken met de texture matrix stack. Toekomstige groepen hoeven de werking hiervan dan niet helemaal opnieuw op internet bij elkaar te sprokkelen.

De texture stack heeft na het initialiseren van OpenGL slechts twee bruikbare coördinaten: de `s` en `t` coördinaten. Deze komen in principe overeen met de `x,y` coördinaten op bijvoorbeeld de model view stack. Om de silhouetten in de ruimte te projecteren is het echter nodig perspectief correcties en rotaties uit te voeren op de texture stack. Hiervoor kan de texture stack als volgt worden geïnitialiseerd:

```
float eyePlaneS[] = { 1.0f, 0.0f, 0.0f, 0.0f };
float eyePlaneT[] = { 0.0f, 1.0f, 0.0f, 0.0f };
float eyePlaneR[] = { 0.0f, 0.0f, 1.0f, 0.0f };
float eyePlaneQ[] = { 0.0f, 0.0f, 0.0f, 1.0f };

glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, (int) GL_EYE_LINEAR);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, (int) GL_EYE_LINEAR);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, (int) GL_EYE_LINEAR);
glTexGeni(GL_Q, GL_TEXTURE_GEN_MODE, (int) GL_EYE_LINEAR);

glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_GEN_R);
glEnable(GL_TEXTURE_GEN_Q);

glTexGenfv(GL_S, GL_EYE_PLANE, eyePlaneS);
glTexGenfv(GL_T, GL_EYE_PLANE, eyePlaneT);
glTexGenfv(GL_R, GL_EYE_PLANE, eyePlaneR);
glTexGenfv(GL_Q, GL_EYE_PLANE, eyePlaneQ);
```

Vervolgens kan de texture stack zich gedragen op een manier die doet denken aan de manier waarop de gehele OpenGL pipeline werkt: object coördinaten worden omgezet in eye coördinaten, eye coördinaten worden omgezet in clip coördinaten en deze vervolgens in window en device coördinaten.

Dit kan als volgt:

$$\mathbf{T} = \mathbf{BS} \cdot \mathbf{P} \cdot \mathbf{M}$$

waarbij **T** de texture matrix is, **BS** een 'bias en scale' matrix, **P** een perspectiefcorrectie matrix en **M** een 'model view' matrix. Dus niet de daadwerkelijke model view en perspecief matrices van OpenGL, maar zelf gedefinieerde.

In code kan dit als volgt (zie ook http://www.opengl.org/resources/code/samples/sig99/advanced99/notes/node80.html):

```
glMatrixMode(GL_TEXTURE);
glPushMatrix();
glLoadIdentity();

// bias & scale
glTranslatef(0.5f,0.5f,0.0f); // bias
glScalef(0.5f,0.5f,1.0f); // scale

// 'perspective matrix'
glFrustum(/* ... */);

// 'modelview matrix'
gluLookAt(/* ... */);
```

```
glPopMatrix();
```

In het huidige prototype zit dus nog een issue, in de `gluLookAt` stap. Een volgende groep kan deze stap verder onderzoeken. De parameters die aan `gluLookAt` moeten worden meegegeven stelt de positie van de camera t.o.v. de slice voor. Deze parameters zijn momenteel incorrect.

# Deel III

# Appendices

Bijlage A

# Wiskundige begrippen en notaties

## A.1 Vectoren

### Notatie

Variabelen die vectors zijn of functies die een vector produceren worden genoteerd met een naar rechts wijzende pijl. Bijvoorbeeld, als $x$ en $y$ integers zijn, dan is

$$\overrightarrow{E(x,y)}$$

een a functie genaamd $E$ van twee integer-variabelen welke een vector oplevert. Een andere notatie is

$$\overrightarrow{E} : \mathbb{Z}^2 \to \mathbb{V}^3$$

waarbij $\mathbb{Z}^2$ 'twee variabelen' van 'de verzameling van alle integers' en $\mathbb{V}^3$ een '3-dimensional vector' betekent.

### Kolomvectoren

Voorbeelden van vectoren worden genoteerd als een matrix met een enkele kolom. Het aantal rijen in de matrix is gelijk aan de dimensie van de vector. Vectoren zijn in dit document dus altijd 'kolomvectoren'. Wat voorbeelden:

$$\overrightarrow{E(3,2)} = \begin{bmatrix} 0.65 \\ 0.19 \\ 0.48 \end{bmatrix} \quad \overrightarrow{E(9,21)} = \begin{bmatrix} 0.53 \\ 0.31 \\ 0.61 \end{bmatrix}$$

## A.2 Matrices

### Notatie

Matrices welke niet ook kolomvectoren zijn worden genoteerd met vet gedrukte variabele namen. Bijvoorbeeld, een $3 \times 2$ matrix $\mathbf{P}$ en een $2 \times 3$ matrix $\mathbf{Q}$:

$$\mathbf{P} = \begin{bmatrix} 2 & 5 \\ 1 & 8 \\ 4 & 17 \end{bmatrix} \quad \mathbf{Q} = \begin{bmatrix} 2 & 5 & 1 \\ 8 & 4 & 17 \end{bmatrix}$$

## Matrixproducten

Omdat alle vectoren in dit document kolomvectoren (en dus ook matrices) zijn, kunnen deze in sommige gevallen worden vermenigvuldigd met andere matrices. Het product van twee matrices is gedefinieerd als het aantal kolommen van de eerste matrix gelijk is aan het aantal rijen in de tweede matrix. Gegeven de voorgaande voorbeelden zijn sommige producten niet gedefinieerd:

$$\begin{aligned}
\overrightarrow{E(3,2)} \cdot \mathbf{P} &\quad \text{niet gedefinieerd} \\
\overrightarrow{E(3,2)} \cdot \mathbf{Q} &\quad \text{niet gedefinieerd} \\
\mathbf{P} \cdot \overrightarrow{E(3,2)} &\quad \text{niet gedefinieerd}
\end{aligned}$$

en is een product wel gedefinieerd:

$$\begin{aligned}
\mathbf{Q} \cdot \overrightarrow{E(3,2)} &= \begin{bmatrix} 2 & 5 & 1 \\ 8 & 4 & 17 \end{bmatrix} \begin{bmatrix} 0.65 \\ 0.19 \\ 0.48 \end{bmatrix} \\
&= \begin{bmatrix} 2 \cdot 0.65 + 5 \cdot 0.19 + 1 \cdot 0.48 \\ 8 \cdot 0.65 + 4 \cdot 0.19 + 17 \cdot 0.48 \end{bmatrix} \\
&= \begin{bmatrix} 2.729999 \\ 14.12 \end{bmatrix}
\end{aligned}$$

## Transponeren

Een '$T$', zoals in $\overrightarrow{E(3,2)}^T$ of $\mathbf{Q}^T$ geeft aan dat een kolomvector of (gewone) matrix getransponeerd dient te worden: een $3\times2$ matrix wordt een $2\times3$ matrix, waarbij de kolommen de nieuwe rijen worden, en de rijen de nieuwe kolommen. Bijvoorbeeld:

$$\overrightarrow{E(3,2)}^T = \begin{bmatrix} 0.65 \\ 0.19 \\ 0.48 \end{bmatrix}^T = \begin{bmatrix} 0.65 & 0.19 & 0.48 \end{bmatrix}$$

$$\mathbf{Q}^T = \begin{bmatrix} 2 & 5 & 1 \\ 8 & 4 & 17 \end{bmatrix}^T = \begin{bmatrix} 2 & 8 \\ 5 & 4 \\ 1 & 17 \end{bmatrix}$$

## Inverse

Een '$-1$', zoals in $\mathbf{R}^{-1}$, geeft aan dat de matrix geïnverteerd dient te worden. Inverses zijn in deze context alleen gedefinieerd als de matrix vierkant is (het aantal rijen is gelijk aan het aantal en kolommen). In dit document wordt alleen de inverse van een $3 \times 3$ matrix gebruikt, welke als volgt kan worden berekend (gekopiëerd uit wikipedia):

$$\mathbf{R}^{-1} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & k \end{bmatrix}^{-1} = \frac{1}{Z} \begin{bmatrix} A & D & G \\ B & E & H \\ C & F & K \end{bmatrix}$$

waarbij

$$Z = a(ek - fh) + b(fg - dk) + c(dh - eg)$$

de determinant is van de matrix. Als $Z \neq 0$, dan is de matrix inverteerbaar, en kunnen de elementen van de matrix aan de rechter zijde worden berekend door

$$\begin{array}{lll} A = (ek - fh) & D = (ch - bk) & G = (bf - ce) \\ B = (fg - dk) & E = (ak - cg) & H = (cd - af) \\ C = (dh - eg) & F = (bg - ah) & K = (ae - bd) \end{array}$$

## A.3 Set-builder notatie

Set-builder notatie wordt in dit document als volgt gebruikt.

$$S \stackrel{\text{def}}{=} \{x : \phi(x)\}$$

betekent dat een verzameling genaamd $S$ wordt gedefinieerd als 'alle $x$ waar voor geldt $\phi(x)$'. Bijvoorbeeld,

$$S_1 \stackrel{\text{def}}{=} \{x \in \mathbb{Z} : x > 42\}$$

is een verzameling genaamd $S_1$ welke alle integers groter dan 42 bevat: $\{43, 44, 45, \ldots\}$.

## Appendix B

# Bibliography


Visual hull (wikipedia). http://en.wikipedia.org/wiki/Visual_hull. (cited on page 14)

L. Guan. Graphcut background subtraction. http://www.cs.unc.edu/~lguan/Research.files/Research.htm. (cited on page 10)

J.-M. Hasenfratz, M. Lapierre, J.-D. Gascuel, and E. Boyer. Real-time capture, reconstruction and insertion into virtual world of human actors. In *Vision, Video and Graphics*, pages 49 – 56. Eurographics, Elsevier, 2003. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.6167&rep=rep1&type=pdf. (cited on pages 13, 15, and 16)

B. Michoud, E. Guillou, and S. Bouakaz. Real-time and markerless 3d human motion capture using multiple views. In *Proceedings of the 2nd conference on Human motion*, pages 88 – 103, Berlin, Heidelberg, 2007. Springer-Verlag. http://liris.cnrs.fr/~bmichoud/publications/humanmotion2007.pdf. (cited on page 13)

B. Petit, J.-D. Lesage, C. Ménier, J. Allard, J.-S. Franco, B. Raffin, E. Boyer, and F. Faure. Multi-camera real-time 3d modeling for telepresence and remote collaboration. *International Journal of Digital Multimedia Broadcasting*, January 2010. http://perception.inrialpes.fr/Publications/2010/PLMAFRBF10. (cited on page 13)

K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: principles and practice of background maintenance. In *Proceedings of the international conference on Computer Vision*, pages 255–261, 1999. (cited on page 7)

P. Turaga, R. Chellappa, V. Subrahmanian, and O. Udrea. Machine recognition of human activities: A survey. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(11):1473 – 1488, November 2008. http://www2.cs.uh.edu/~shah/courses/cosc7373/files/activityrecnsurvey.pdf. (cited on page 7)